ME5406 Project 1 Report

Semester 2 AY2023/2024

A0216988J
Chin Zheng Hao

1. Introduction

In this project, we were tasked with finding the optimal policy for a robot traversing a 2D grid to reach a goal (a frisbee), upon which it will receive a reward of 0. This 2D grid has regions with thin ice, and if the robot traverses over these, it will fall through and die (receiving a reward of –1). In all other locations, the robot receives a reward of 0.

3 algorithms: First Visit Monte Carlo (FVMC), Q Learning, and SARSA were implemented. This report will describe, discuss and compare the performances of each of these algorithms. For the sake of brevity, the details of the algorithms' implementation are left out. Each implementation and relevant comments are included in the source code that accompanies this report

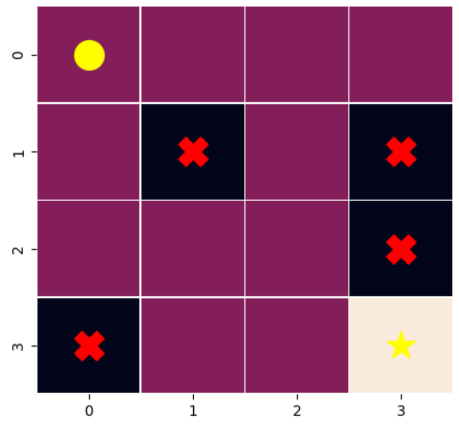
2. State Transition Model

We follow the state transition model provided by the assignment brief. The robot can move either up, down left or right, and is confined to the grid. This means that actions that cause it to move out of the grid result in no movement.

3. 2D Grid Environment

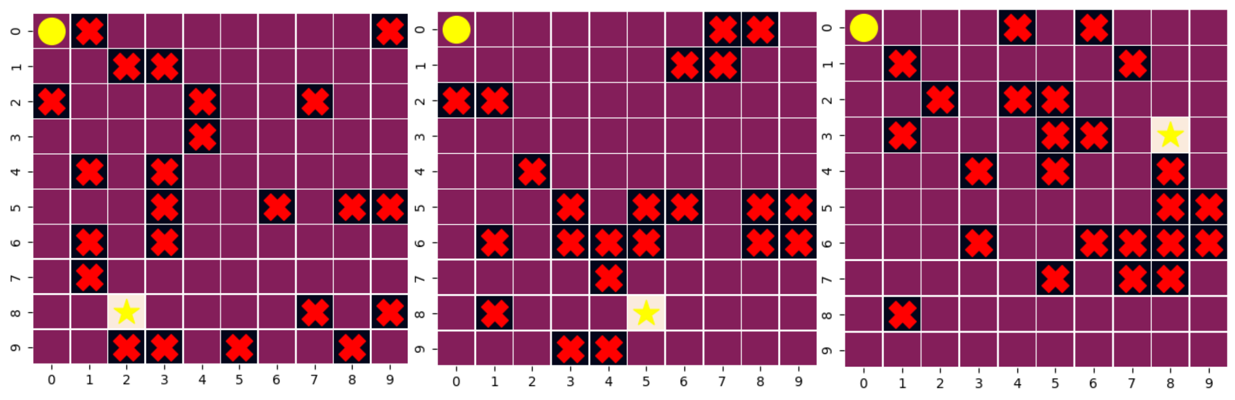
Representations of the grid environments used are shown below. Here, each cell represents a state (position on the 2D grid). The crosses represent the thin ice regions, the yellow star represents the goal position, and the yellow circle represents the origin.

In this project, we use 2 types of environments for evaluating the algorithms. The first is the basic 4x4 environment provided in the assignment brief. We shall refer to this environment as the basic environment.



The basic environment

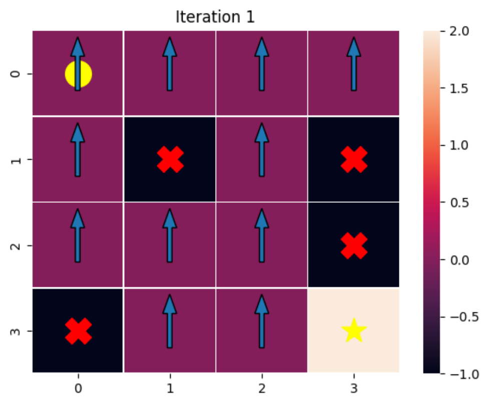
The second type is the larger 10x10 environments detailed in part 2 of the assignment. While we can generate the second environment type randomly, we use the 3 specific instances of them shown below to evaluate the algorithms. We shall refer to these as advanced environments 1 to 3.


Advanced environments 1, 2 and 3 respectively from left to right

4. Evaluation Metrics

We will be evaluating each algorithm using the following:

1. Average reward (window size of 100) against iteration number
    a. This will also serve as a proxy for success rate, since we do not penalize spending more time traversing the grid, and episodes that reach the goal will have the same reward regardless of the path taken.
2. Steps per episode against iteration number
    a. This allows us to track the convergence of each algorithm and the path length of the policy in each iteration.
3. Training time
4. Q Table progress
    a. Since these algorithms all use a Q Table for storing the value of each state action pair, we can use the Q Table to visualize how the policy evolves over time
    b. The arrows represent the action with the highest value, and each state's value is reflected by the cell color.
    c. Running the code accompanying this report will produce snapshots of the Q Table at different iterations, and a gif that reflects how it evolves with the iterations.
    d. This can also be used to determine how much of the state space the robot has explored, as properly explored states should 'point' towards a path that leads to the goal state.

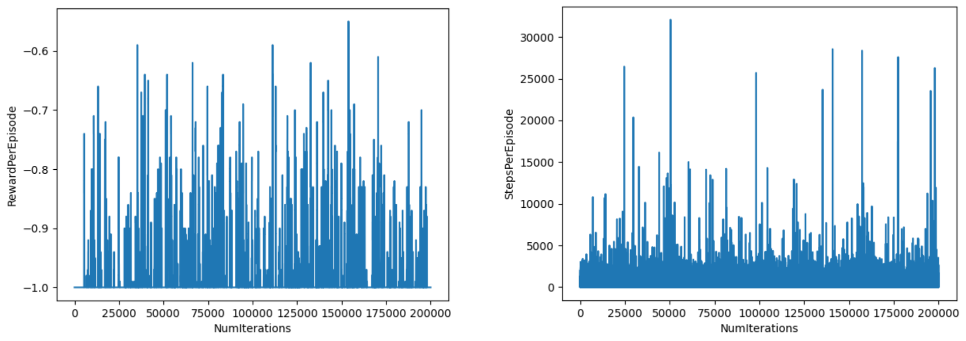


5. First Visit Monte Carlo (FVMC)

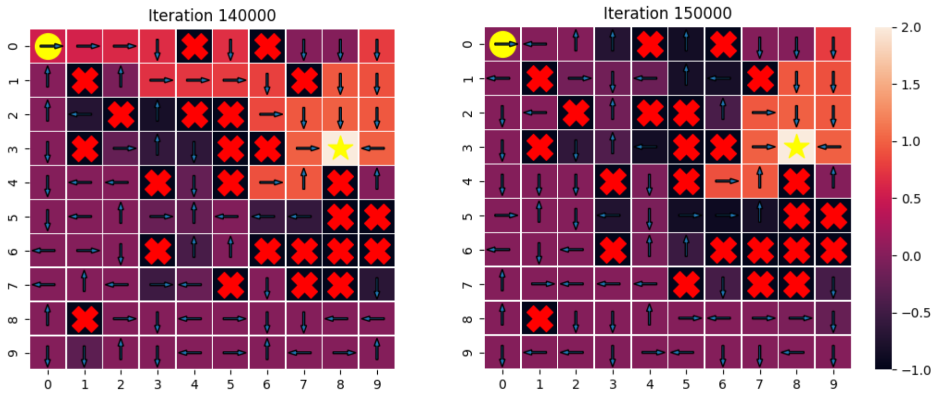
Unmodified Implementation

The unmodified FVMC algorithm was ran on the 4 test environments detailed in section 3 using an epsilon greedy policy with epsilon set to 0.1, a batch size of 2, and discounting factor 0.95. In general, the algorithm struggles to converge.

The algorithm tends to find a policy that leads it to the goal, but the random action due to the epsilon greedy policy results in the robot reaching a thin ice region while following the policy it found. This causes the state action pairs encountered throughout the episode to be estimated as having negative value despite leading to the goal. The update rule then overwrites the Q Table entries with these erroneous state action pair value estimates.

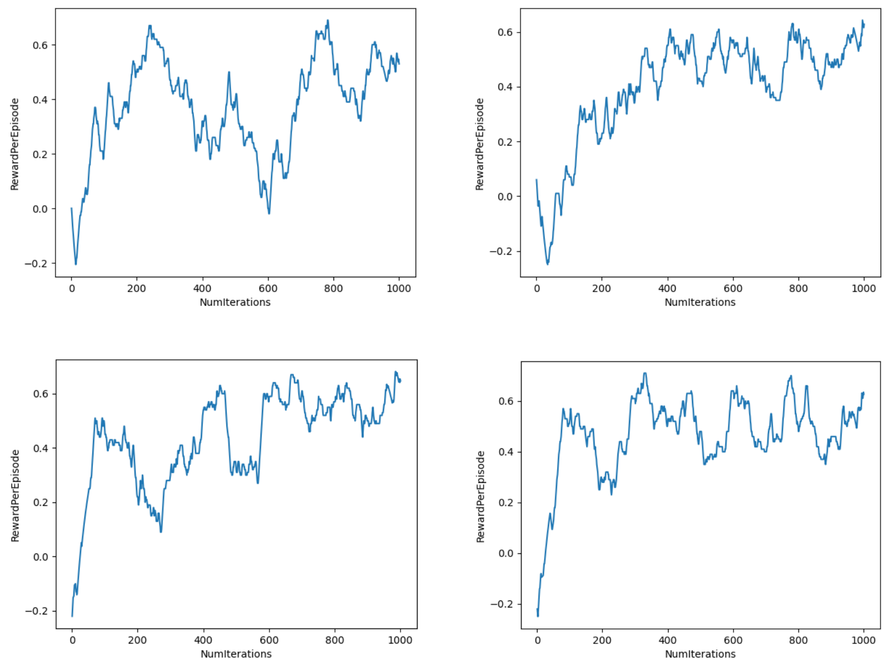


Graphs of rewards per episode and steps per episode against the number of iterations, for FVMC running in advanced environment 3.



FVMC Finds the optimal policy for advanced environment 3 in iteration 140000, but the epsilon greedy policy and update rule causes it to not be able to converge, as seen in iteration 150000.

This problem is less apparent when studying the performance of FVMC in the basic environment. The graphs below show the average reward per episode across 4 consecutive runs in the basic environment. While the frequent dip in average rewards is indicative of the problem described above, it is much less significant.



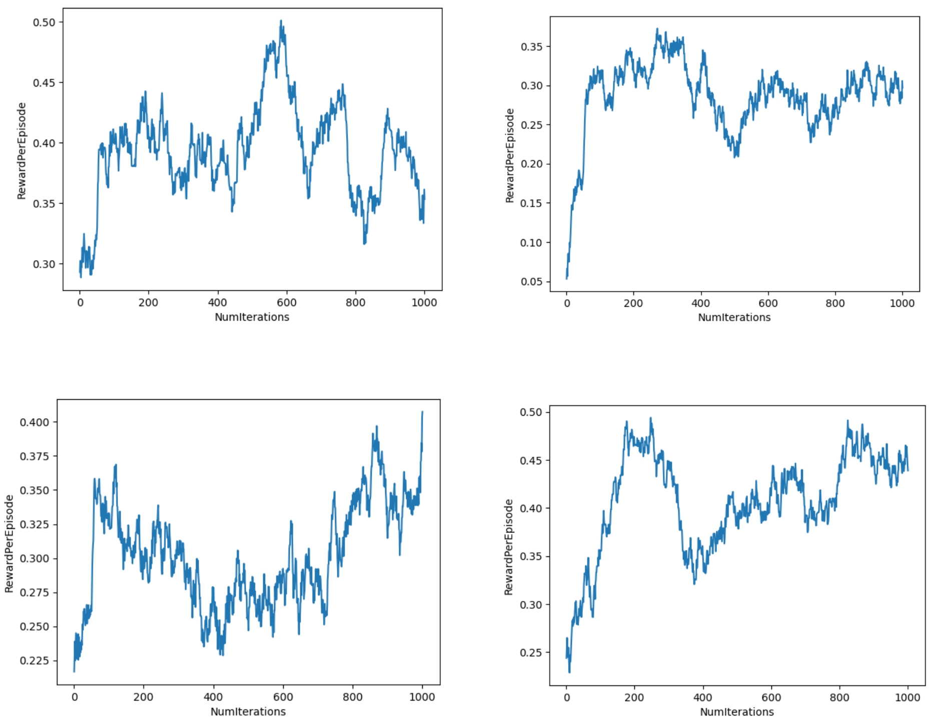
Graphs of rewards per episode against iterations for FVMC running in the basic environment

One reason why this failure to converge is more apparent in larger environments is that in larger environments, there are more steps to take, and hence more chances to make wrong moves that lead to terminating with a negative reward given the epsilon greedy policy. This means that a robot following a policy that will lead it to the goal in the larger environment is more likely to fail and terminate with a negative reward and an erroneous update step than the same robot with an equally good policy in a smaller environment.

Experiments: Increasing Batch Size

One would expect that the above problem can be fixed by choosing a larger batch size. By averaging across batches, the robot will still be able to reach the goal on average, and the state action pair value estimates should reflect that when updating the Q Table with a large batch size. In general, increasing the batch size produces an upwards trend in

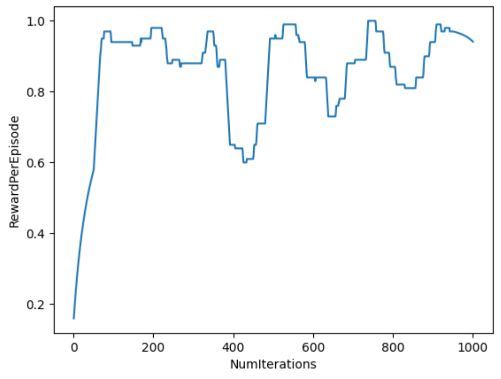
Interestingly, running the FVMC algorithm with a batch size of 500 on the basic environment did not result in much change. This could be attributed to the fact that with a larger batch size, the FVMC algorithm will likely encounter a larger set of states that are not part of the path produced by a goal reaching policy per batch. This could result in more updates that increase the value of poorer state action pairs above that of better state action pairs (given the same state). The result is a noisy update step, producing the results seen below.

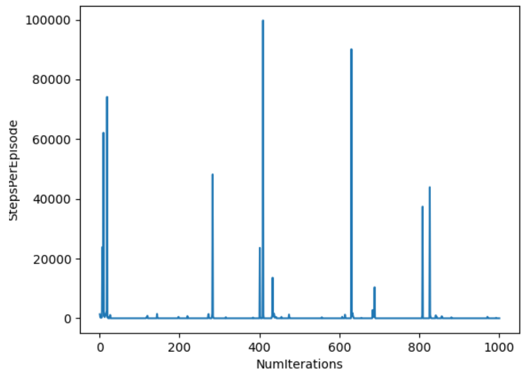


Graphs of rewards per episode against the number of iterations, for FVMC running in the basic environment with a batch size of 500

Experiments: Reducing Epsilon

Reducing epsilon should reduce the chance of the robot making the ‘wrong’ choice due to a chance random action. This indeed does seem to alleviate the problem, as shown when running FVMC in a basic environment with a batch size of 2, and an epsilon of 0.01.

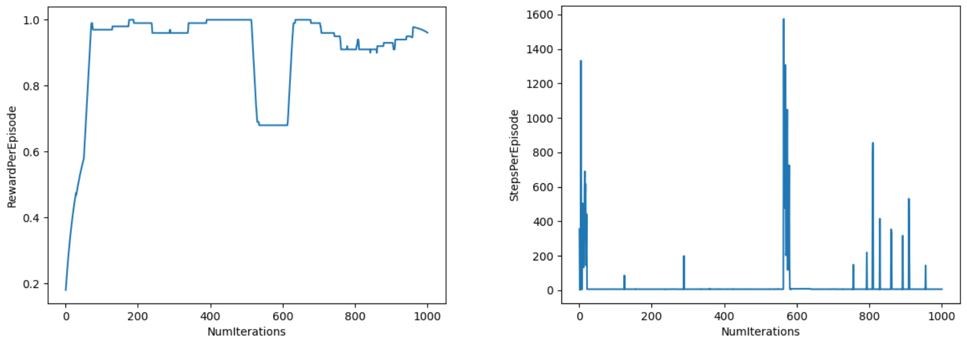




However, this did not fully fix the problem, as seen by the fluctuating average reward values. Additionally, the number of steps per episode explodes due to the robot repeatedly choosing ‘useless actions’ (actions that result in no movement, as the robot is at the edge of the grid) many times before a random move is executed to bring the robot out of that state.

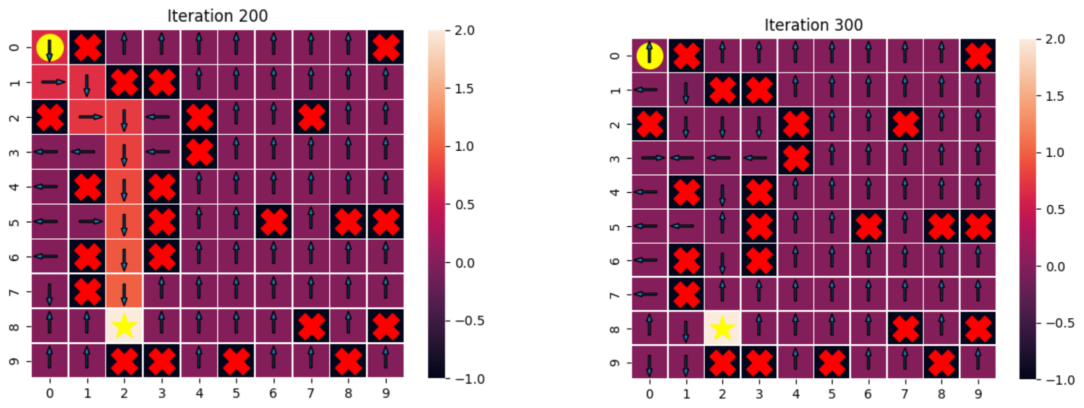
Experiments: Culling Useless Actions

We implemented a check that removes actions that result in no movement from consideration. This fixed the ‘exploding steps’ problem to a degree and resulted in a shorter training time per batch as the robot is no longer caught in situations where it repeatedly chooses a useless action.



Steps per iteration decreased due to culling useless actions

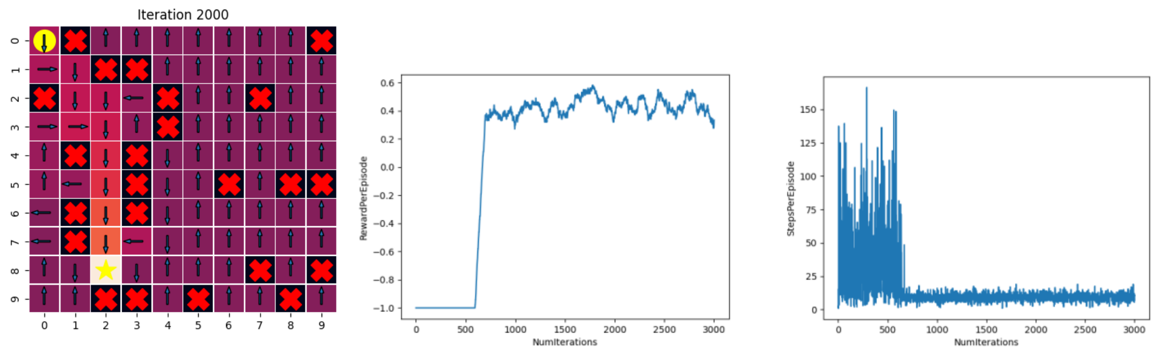
Unfortunately, the combined effect of culling useless actions and reducing epsilon to 0.001 still resulted in failure to converge in larger maps. Reducing epsilon further dramatically increased the training time, and would probably not completely solve the problem as given enough iterations, an erroneous update step is still going to happen



The same failure to converge when running FVMC on larger grids like advanced environment 1

Experiments: Modifying the update rule

Instead of overwriting the Q Table value for a given state action pair (B) with its value as evaluated by a particular batch (α), we can instead ‘shift’ the Q Table value towards it using the update rule B = B + μ × (α - B), where 0 < μ <= 1. Setting μ to 1 gives us our original update rule. This means that the erroneous updates do not completely overwrite the Q Table values, solving the failure to converge problem. This modification allows FVMC to successfully converge on solutions for advanced environments 1.



FVMC with updated rule converging on a solution for advanced environment 1

However, FVMC with modified update rule still struggles with advanced environments 2 and 3. Each batch takes very long, and we still do not converge.

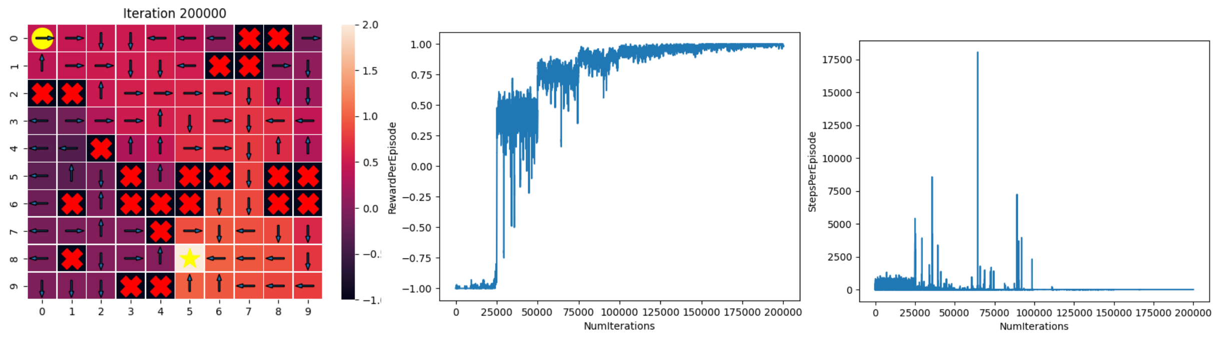
One potential reason is that in advanced environment 1, the 'explorable' region surrounding the initial position of the robot is enclosed by many thin ice regions and is very small. Hence, even with a low epsilon value, the robot is exploring enough to find the solution and converge to it. This same reason also causes episodes in advanced environments 2 and 3 to last longer, resulting in longer iteration times per batch.

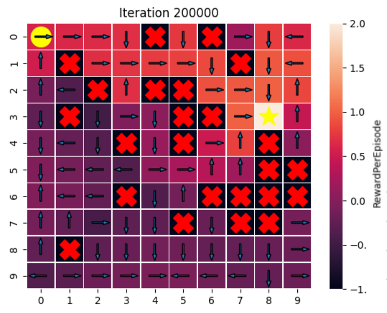
Experiments: Update pruning and epsilon decay

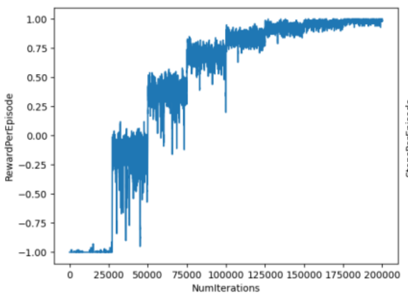
Update pruning cuts off the update step of FVMC for steps that are too early. As the length of an episode increases, the magnitude of updates earlier in the episode becomes more and more insignificant due to the discounting factor. Since the reward is only given in the terminal state, we can cut off the update step after the compounded discounting factor has reached 0.01. This allows the update step to run quicker without much loss.

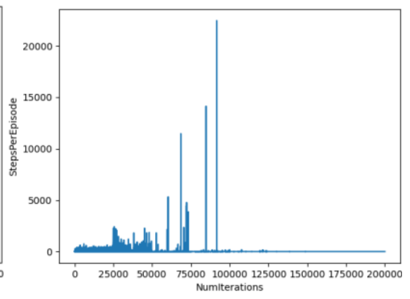
Epsilon decay allows us to start off with a large value of epsilon (0.5) and decrease it as the batch number increases. This way, the robot will explore more towards the start of training and converge on a solution as the iteration number increases. We implement this by halving epsilon every 25000 batches.

Running the with all the modifications described thus far allows it to converge to a solution for advanced environments 2 and 3 as well. This version of FVMC converges at around iteration 25000-50000 for most randomly generated 10x10 grid environments. This takes around a minute. In almost all cases, inspection of the Q Table images show that the grid is well explored.







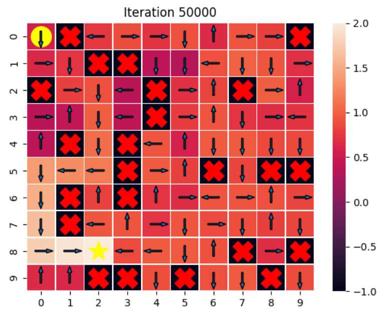


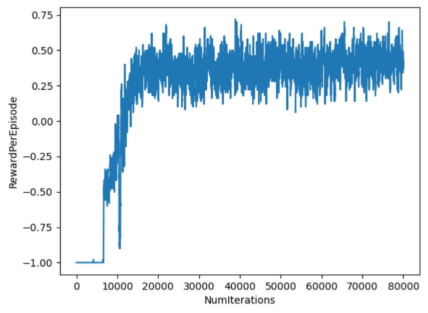
SARSA

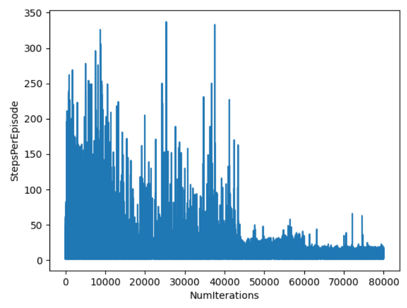
Unmodified Implementation

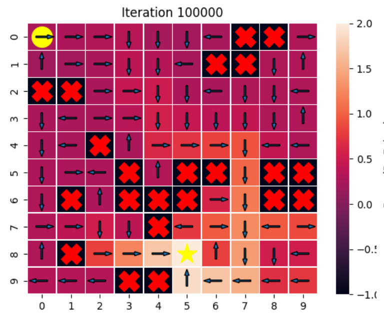
The unmodified implementation of SARSA with an epsilon greedy policy of (epsilon = 0.1), a learning rate of 0.0005, and a discounting factor of 0.95 converges to solutions for the basic environment and advanced environments 1, 2 and 3 (as well as other randomly generated environments).

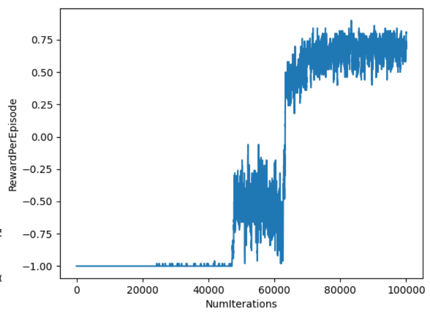
SARSA with these settings converges at around iteration 15000-50000 for most randomly generated 10x10 grid environments. This takes around 30 seconds. In most cases, inspection of the Q Table images show that the grid is well explored.

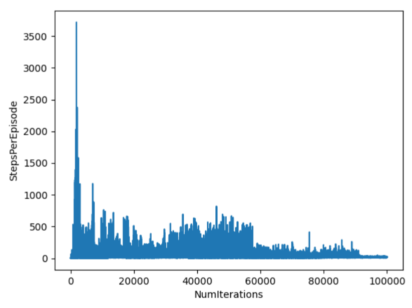


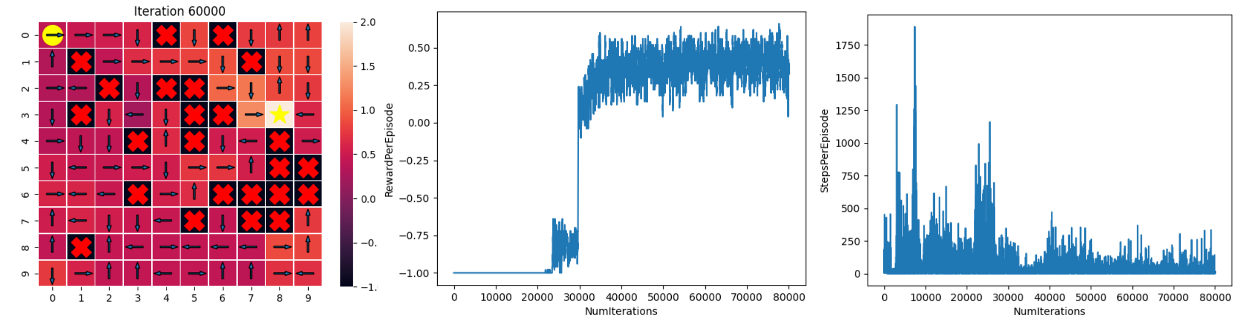












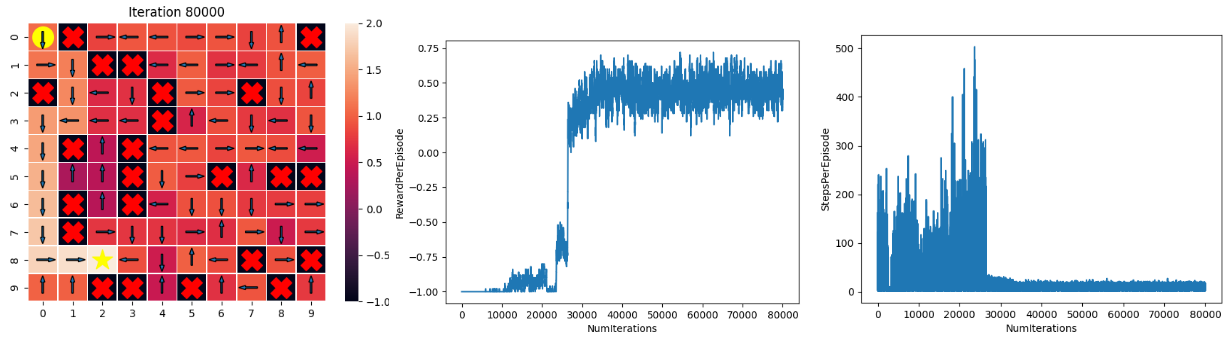
Results of SARSA executed on advanced environments 1,2 and 3

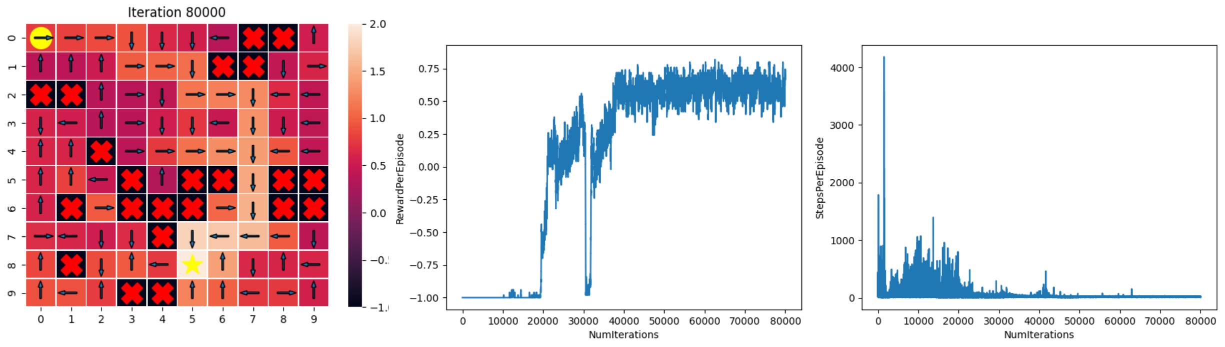
Q Learning

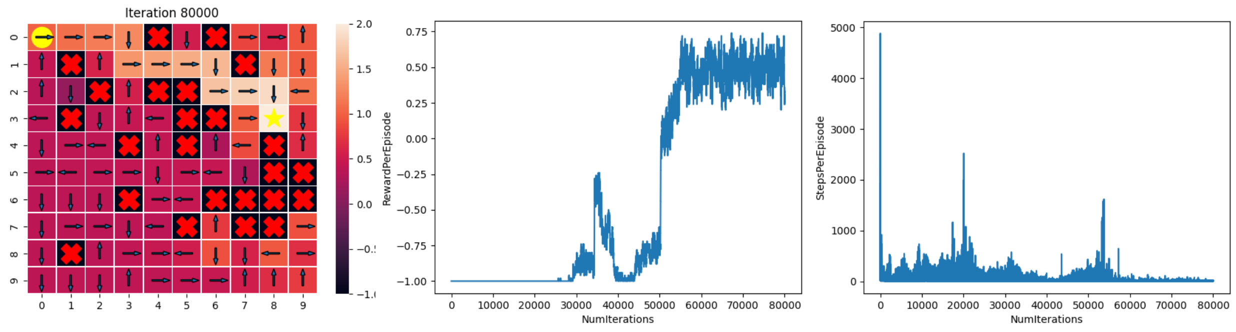
Unmodified Implementation

The unmodified implementation of Q Learning with an epsilon greedy policy of (epsilon = 0.1), a learning rate of 0.0005 and a discounting factor of 0.95 converges to solutions for the basic environment and advanced environments 1, 2 and 3 (as well as other randomly generated environments).

Q Learning with these settings converges at around iteration 30000-50000 for most randomly generated 10x10 grid environments. This takes around 30 seconds. In most cases, inspection of the Q Table images show that the grid is well explored







Results of Q Learning executed on advanced environments 1,2 and 3

Comparisons and Conclusions

The table below summarizes the performance of each algorithm (with the same hyperparameters as described above) on a grid of size 10x10 (including the 3 advanced environments). Despite modified FVMC, SARSA and Q Learning all taking roughly the same number of iterations to converge, the time taken per iteration is longer for modified FVMC, resulting in it having a slightly longer training time on average.

The epsilon decay of modified FVMC allows it to start with a larger value of epsilon and explore more states, explaining the higher degree of exploration observed when using it.

	FVMC	Modified FVMC	SARSA	Q Learning
Iterations to convergence	NIL	25000-50000	15000-50000	30000-50000
Degree of exploration	Poor	Very good	Good	Good
Training time	> 30 mins for some scenarios	~1 min	~30s	~30s

The comparisons between the algorithms produced in this report are not fully comprehensive, as the same algorithm can perform differently in different scenarios and with different hyperparameters. Running SARSA with a different learning rate may dramatically decrease training time or result in no convergence.

Additionally, not all configurations of each algorithm are evaluated. For example, running SARSA with epsilon decay should result in a better degree of exploration for SARSA, although this comparison is omitted in this report.

Overall, we can conclude that the 3 algorithms implemented, with the hyperparameters described in this report, are effective in finding an optimal policy to reach the goal state.